05

Hibernate y JPA

Transcripción

[00:00] Hola. Ya hemos discutido un poco sobre JDBC y los principales problemas que los desarrolladores comenzaron identificar a medida que creaban sus aplicaciones. La existencia de estos problemas fue precisamente lo que motivó a las personas a buscar alternativas y tecnologías que fueran más sencillas para realizar la conexión con la base de datos.

[00:16] Entre estas tecnologías había una biblioteca que se hizo muy famosa llamado Hibernate. Hibernate fue lanzada en 2001 y creada por Gavin King. La idea era precisamente intentar simplificar JDBC. Suponiendo que tenemos una aplicación web o de escritorio necesitamos acceder a la base de datos, pero no queremos utilizar JDBC porque el código es muy detallado, complejo y está muy acoplado a la base de datos.

[00:42] Gavin King comenzó a pensar en una forma de simplificar el código y creo esta biblioteca llamada Hibernate, y en 2001 fue lanzada al mercado para realizar la persistencia de bases de datos como alternativa a JDBC y EJB 2. En ese momento existía la versión 2.0 de EJB 2 que era una tecnología muy complicada para trabajar.

[01:03] La idea de EJB era simplificar el acceso remoto, tener una aplicación distribuida cliente-servidor y también simplificar algunos detalles de infraestructura como control transaccional, seguridad y otros.

[01:16] También estaba la parte de persistencia con JB, pero esta utilizaba JDBC, un modelo un poco más complejo que favorecía la removilidad,

entonces cualquier llamada que se hacía era una llamada remota. Eso tenía un costo de rendimiento Y de todos modos surgieron varios problemas que también generaron varios patrones para resolver esos problemas en EJB2 en la parte de persistencias.

[01:39] Eso también motivó a Gavin King al crear Hibernate. Más tarde Red Hat contrató a Gavin King para continuar trabajando en Hibernate. Así Hibernate pasó a ser una tecnología que pertenece a JBoss de Red Hat.

[01:53] Gavin King trabajo en JBoss y continuó con hibernate y otros proyectos. Con el paso del tiempo aparecieron nuevas versiones de Hibernate, este se hizo famoso en todo el mundo y todos los que trabajaban con Java querían utilizar Hibernate en sus proyectos para no tener que utilizar el EJB 2 ni JDBC.

[02:12] De todos modos se convirtió en un proyecto súper popular y fue evolucionando y luego aparecieron versiones con nuevas características. Como era una biblioteca de mercado también surgieron competidores, por lo tanto estamos hablando una biblioteca que se hizo famosa, popular y de este modo surgieron bibliotecas que comenzaron a copiarla pero haciéndolo de una manera diferente.

[02:34] Esto generó un viejo problema. Imagina que como desarrolladores estamos utilizando una biblioteca y queremos pasar a usar, otra una biblioteca de mercado más económica o con mejores características. Para ello no se trata solo de cambiar los archivos jar o las dependencias del proyecto, esto tendría un impacto considerable en el código.

[02:53] En todos los lugares donde estábamos utilizando Hibernate, va a ser necesario cambiar las características del código. Para esta nueva biblioteca ya que hay otras clases, otras importaciones y si nuestro proyecto fuese muy grande y complejo lo pensaríamos dos veces antes de cambiar la biblioteca o la librería.

[03:14] A Oracle esto no le pareció una buena idea porque esto significa que trabaja con un único proveedor. Posteriormente se creó una estandarización de la librería para Java, va a partir del modelo de persistencia que pasó conocerse como ORM por sus siglas Object Relational Mapping, con la intención de hacer el mapeo y así el puente entre el mundo de las aplicaciones orientadas a objetos y las bases de datos relacionales.

[03:40] Existen estos dos mundos diferentes y la clase DAO era un poco compleja porque estábamos uniendo el mundo orientado a objetos con el mundo de las bases de datos relacionales. Así Java creó una especificación llamada JPA, API de persistencia de Java, que es la especificación para estandarizar el mapeo relacional de objetos en el mundo Java.

[04:01] Con JPA crea un estándar para que no seamos rehenes de una única biblioteca. Los frameworks y las bibliotecas comenzaron a implementar JPA en el código en lugar de importar las clases e interfaces de Hibernate. Comenzamos a usar JPA que es la especificación, por lo tanto la librería se convirtió una implementación.

[04:22] Y para cambiar de implementación, solo necesitamos cambiarlo en jar y una u otra configuración. Pero el código en sí permanecía intacto y sin cambios, ya que no dependía de una implementación, sino de esta especificación JPA.

[04:35] JPA no se lanzó sino hasta 2006, por lo que de 2001 hasta 2006 necesitamos usar Hibernate o sus competidores sin la especificación JPA, que era la que nos orientaba el modo correcto de usar los códigos.

[04:49] Hibernate evolucionó surgieron nuevas funciones y luego se incorporado a la versión 2.0 de JPA lanzada en 2009. Hibernate en 2010 un año después lanzó la versión 3.5.0 que era compatible con JPA 2.0.

[05:07] Entonces sí quisiéramos usar JPA 2.0, podríamos usar Hibernate como implementación, y si luego quisiéramos cambiar a otras implementaciones

sería bastante fácil y nos afectaría todo el proyecto. Esta es la gran ventaja de usar la especificación en lugar de usar directamente la implementación.

[05:25] En el mercado nos quedamos con algo como este diagrama donde tenemos a JPA arriba, que es la especificación y también tenemos varias implementaciones como Hibernate, EclipseLink y OpenJPA, entre otras implementaciones del mercado. Estas son las más populares y las primeras que se usan en el mercado.

[05:46] Para trabajar con JPA, tenemos que elegir una de estas implementaciones, es decir no podemos utilizar JPA puro, porque JPA es una capa o una abstracción únicamente. Necesitamos a alguien que implemente los detalles con lo que las bibliotecas como hibernate o EclipseLink hacen ese trabajo.

[06:06] Como Hibernate fue la primera biblioteca, fue entonces la que inició este movimiento fue lo que terminó convirtiéndose en la biblioteca estándar y la más popular, la implementación de JPA por excelencia, pero nada nos impide utilizar otras implementaciones si todo se orienta con JPA.

[06:26] Todos deben cumplir con lo que se describe en la especificación. Además podemos cambiar de una implementación a otra y todo lo que está en la especificación funcionará correctamente. Solo tendremos problema si estamos utilizando algo que sea específico de la implementación.

[06:41] Hibernate a veces hace eso. Sigue las reglas de JPA pero tiene algunos detalles bien característicos que son específicos para Hibernate. Si los usamos, está bien, pero tendremos este problema, que a la hora de nosotros movernos de una implementación. A otra, vamos a perder esta esta funcionalidad o esta nueva característica.

[07:05] Entonces es recomendable mantenernos dentro del patrón de JPA. Entonces, por ejemplo, si queremos cambiar de Hibernate a EclipseLink,

vamos a perder esa funcionalidad. Tenemos que tener cuidado y evaluar si está dependencia, esta funcionalidad realmente vale la pena.

[07:20] Eclipselink es la implementación de referencia de JPA. Cada vez que aparece una nueva versión de JPA, EclipseLink ya la está implementando porque es donde se hacen las pruebas y sale con la nueva versión de JPA.

[07:33] Es la implementación de referencia, sin embargo Hibernate al ser la principal y la más popular del mercado es la que tiene mayor demanda. Por eso en este curso trabajaremos con Hibernate como una implementación de JPA.

[07:48] Vamos a finalizar esta parte de la motivación y nuestro objetivo aquí era solamente discutir un poco y entender qué es JPA, por qué se creó, de que nació, qué es Hibernate, cuál es la diferencia entre hibernate JPA.

[08:04] Y ahora que hemos visto esta característica, esta breve historia, vamos a pasar al siguiente video donde vamos a comenzar con la parte práctica nos vemos en el siguiente video.